Proyecto Fin de Carrera
Ingeniería de Telecomunicación

ParkApp: Aplicación Móvil para Android de Búsqueda de Aparcamiento

Autor: Francisco Javier García Jurado
Tutor: Federico José Barrero García

Dep. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2015

Proyecto Fin de Carrera
Ingeniería de Telecomunicación

ParkApp: Aplicación móvil para Android de búsqueda de aparcamiento

Autor:
Francisco Javier García Jurado

Tutor:
Federico José Barrero García
Profesor titular

Dep. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2015

Proyecto Fin de Carrera: ParkApp: Aplicación móvil para Android de búsqueda de aparcamiento

Autor: Francisco Javier García Jurado

Tutor: Federico José Barrero García

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2015

El Secretario del Tribunal

A mi familia, y en especial
a mis abuelos.

Agradecimientos

Me gustaría recordar, en este instante en el que va a terminar una importante etapa de mi vida, a los que siempre estuvieron ahí apoyándome.

En primer lugar, a mis padres y a mi familia en general. Una especial mención a mis abuelos que siempre tuvieron la convicción de que lograría acabar la carrera pero que por desgracia ya no se encuentran aquí para compartirlo con ellos.

En segundo lugar, a mi tutor, ya que me dio la posibilidad de realizar el desarrollo de la aplicación durante mi estancia en el Reino Unido y siempre respondió mis dudas y consultas de forma casi inmediata.

Por último, también me gustaría darle las gracias a los compañeros que después de estos años se han convertido en mis amigos, tanto en Sevilla, como en Florencia.

Francisco Javier García Jurado

Sevilla, 2015

Resumen

El objetivo de este proyecto fin de carrera es el de desarrollar una versión preliminar de una aplicación móvil para la búsqueda de aparcamiento denominada ParkApp, para el sistema Android.

El principal objetivo de ParkApp es ofrecer al usuario una lista con los distintos parkings públicos que se encuentren dentro del radio indicado al realizar la búsqueda.

Se estudiarán las alternativas que existen en el mercado a ParkApp y se detallarán las diversas herramientas y tecnologías usadas para el desarrollo de la aplicación. También se analizará el resultado del proyecto mediante capturas de pantallas obtenidas y directamente sobre el código de la aplicación.

Por último, se desarrolla un modelo de negocio y se expone el posible futuro en el mercado para ParkApp.

Después de varios meses en el desarrollo del proyecto, se ha obtenido un prototipo funcional para Android. Aunque se trata de una versión preliminar, que aún debe implementar funcionalidades adicionales para poder competir en el mercado, el objetivo principal del proyecto, que era el desarrollo de una versión preliminar, se ha cumplido.

Abstract

The objective of this final degree project is to develop a preliminary version of ParkApp, a mobile application for the search of parking lots, for Android system.

The main purpose of ParkApp is to provide a list of public parking lots located within the radius set by its users.

ParkApp's alternatives, which already exist in the market, will be studied and the diverse tools and technologies used for the application's development will be detailed. In addition, the project's result will be analyzed through screenshots and directly in the application's code.

Finally, a business model will be developed and the potential of ParkApp's future in the market will be exposed.

Having worked in the project's development for several months, a functional prototype for Android has been obtained. And the main objective of the project, develop a preliminary version of ParkApp, has been accomplished although additional functionalities still need to be implemented in order to make the app a competitive product in the market.

Índice

Agradecimientos **ix**

Resumen **xi**

Abstract **xiii**

Índice **xv**

Índice de Tablas **xviii**

Índice de Figuras **xx**

1 Introducción **1**

2 Objetivo y Alcance del Proyecto **2**

3 Estado del Arte **3**

3.1 Parkopedia Estacionamiento *3*

3.2 ParkMe Parking *4*

3.3 Parking *4*

3.4 m.Parking *5*

3.5 Parking Manijak *5*

3.6 apparkB *6*

4 Tecnologías Empleadas **7**

4.1 Entorno de desarrollo *7*

4.2 Tecnologías empleadas *8*

4.2.1 Android 8

4.2.2 Servidor central y bases de datos del servidor 8

4.3 Alternativas existentes a las tecnologías empleadas *9*

4.3.1 Alternativas a Android 9

4.3.2 Alternativas a Apache 11

4.3.3 Alternativas a MySQL 11

5 Desarrollo de ParkApp **12**

5.1 Introducción *12*

5.2 Esquema básico de funcionamiento del sistema *12*

5.3 Diagramas de casos de uso *13*

5.3.1 Actores 13

5.3.2 Diagramas de casos de uso del usuario 13

5.4 Diagramas de clases *14*

5.4.1 Clase MainActivity 14

5.4.2 Clase Descarga 16

5.4.3 Clase Lista 16

5.4.4 Clase Confg y Preferencias 17

5.4.5 Clase Info 18

5.5 Diseños de bases locales *18*

5.6 Evaluación y pruebas *20*

5.6.1 Prueba 1: Crear marcador en el mapa 20

5.6.2 Prueba 2: Dibujar círculo con la distancia 21
5.6.3 Prueba 3: Comprobar distancia entre marcadores 21
5.6.4 Prueba 4: Cambiar vista del mapa 22
5.6.5 Prueba 5: Mostrar mensaje si se introduce una distancia errónea 23
5.6.6 Prueba 6: Cambiar parámetros correctamente 23
5.6.7 Prueba 7: Mostrar lista de ejemplo 24

6 Plan de Implementación de ParkApp 25
6.1 Concepto de negocio 25
6.2 Presentación de la empresa 26
6.3 Visión, misión y objetivos 26

7 Coste del Proyecto 27
7.1 Coste de recursos humanos 27
7.2 Costes adicionales 28
7.3 Coste total del proyecto 28

8 Conclusiones 29

Referencias 31

Glosario 34

Anexo A 36
A.1 Clase MainActivity 36
A.2 Clase Descarga 41
A.3 Clases Lista, Parking y ParkingAdapter 41
A.3.1 Clase Lista 41
A.3.2 Clase Parking 42
A.3.3 Clase ParkingAdapter 43
A.4 Clases Conf y Preferencias 44
A.4.1 Clase Conf 44
A.4.2 Clase Preferencias 44
A.5 Clase Info 45

ÍNDICE DE TABLAS

Tabla 1: Listado de pruebas 20
Tabla 2: Coste de recursos humanos 27
Tabla 3: Costes adicionales 28
Tabla 4: Coste total del proyecto 28

ÍNDICE DE FIGURAS

Figura 1: Información sobre Parkopedia Estacionamiento 3
Figura 2: Información sobre ParkMe Parking 4
Figura 3: Información sobre Parking 4
Figura 4: Información sobre m.Parking 5
Figura 5: Información sobre Parking Manijak 5
Figura 6: Información sobre apparkB 6
Figura 7: Gráfica sobre la venta de *smartphones* según su sistema operativo 10
Figura 8: Esquema básico funcionamiento del sistema 12
Figura 9: Actores del sistema 13
Figura 10: Diagrama de casos de uso del usuario 13
Figura 11: Clases que componen ParkApp 14
Figura 12: Captura de la pantalla principal con el menú desplegado 15
Figura 13: Fragmento de código del *inflater* del menú 15
Figura 14: Fragmento de código del menú 15
Figura 15: Capturas de la animación 16
Figura 16: Fragmento de código del tipo Parking 16
Figura 17: Captura de pantalla de la lista 17
Figura 18: Capturas de la pantalla de configuración 17
Figura 19: Fragmento de código de la clase Preferencias 18
Figura 20: Captura de la pantalla de información 18
Figura 21: Fragmento código aplicación Base de Datos 19
Figura 22: Captura de la pantalla de inserción 19
Figura 23: Fragmento código marcador 20
Figura 24: Captura de la pantalla principal con el marcador 20
Figura 25: Fragmento código círculo 21
Figura 26: Captura de la pantalla principal con el c´rculo 21
Figura 27: Fragmento código distncia entre marcadores 21
Figura 28: Capturas de pantalla distancia entre marcadores 22
Figura 29: Capturas de pantalla vistas del mapa 22
Figura 30: Capturas de pantalla cambio de idioma 23
Figura 31: Captura de pantalla de la lista 24
Figura 32: Diagrama de Gantt 28

1 INTRODUCCIÓN

Una aplicación móvil o app es una aplicación informática diseñada para ser ejecutada en teléfonos inteligentes, tabletas y otros dispositivos móviles. Por lo general se encuentran disponibles a través de plataformas de distribución operadas por las compañías propietarias de los sistemas operativos móviles, en nuestro caso, la Google Play de Android. Las aplicaciones móviles son creadas para dar un acceso rápido y eficaz a todo tipo de necesidades de los usuarios, desde apuntar una nota o escuchar música hasta realizar una operación bancaria o acceder a tu correo electrónico, en nuestro caso, buscar aparcamiento cercano.

Antes de prestar atención en la importancia que tienen hoy en día las aplicaciones móviles se va a hablar de su increíble y reciente historia. Tomando como referencia la infografía sobre la historia de las tiendas de aplicaciones móviles resulta fascinante darse cuenta que el 10 de julio de 2008 se lanzó la App Store, tienda de aplicaciones móviles de Apple, con 500 aplicaciones móviles y el 22 de octubre del mismo año se lanzó la Android Market, tienda de aplicaciones móviles de Android, con tan solo 50 aplicaciones. El 29 de noviembre de 2008, cuatro meses más tarde de su creación, la App Store ya contaba con 10.000 aplicaciones y en enero de 2009 habían sido descargadas más de 500 millones de aplicaciones. Al mirar las cifras del siguiente año, 2010, la App Store contenía 225.000 aplicaciones y 5 billones de descargas, y la Android Market 80.000 aplicaciones y 1 billón de descargas. Por último, la infografía muestra los datos de 2011 con 500.000 aplicaciones y 10 billones de descargas en la App Store, y 250.000 aplicaciones y 6 billones de descargas en la Android Market.

Para hacerse una idea del brutal crecimiento que se ha producido, de los datos anteriores se desprende que en tan solo dos años, el número de aplicaciones en la App Store se multiplicó por 1.000 y en la Android Market se multiplicó por 5.000, algo realmente inverosímil.

Para hablar del interés y la importancia actual de las aplicaciones móviles hay que centrarse en un estudio de 2013 que hace una comparación con las cifras que habrá en 2017. Según el estudio, en 2012 hay 1,2 billones de usuarios de aplicaciones móviles, pero en 2017 habrá 4,4 billones; esto supone un aumento de un 366% en 5 años. Otro dato importante del estudio es el número de aplicaciones descargadas cada año, mientras que en 2012 se descargaron 45,6 billones de aplicaciones, en 2017 se descargarán 309,6 billones; un aumento de un 679%, algo verdaderamente escalofriante. Por último, y puesto que la aplicación objetivo de este proyecto es para el sistema Android, se pueden extraer del estudio dos datos interesantes; en el 2013 el 59% de las aplicaciones descargadas fue desde un dispositivo Android y la cuota de mercado de Android era del 76%. Teniendo en cuenta los datos del segundo trimestre de 2014, el sistema operativo Android ya tiene una cuota de mercado del 84,6%. También es importante para la aplicación móvil objetivo, puesto que se encarga de buscar aparcamiento, remarcar que el 25 de junio de 2014 Google anunció Android Auto, que será la versión de su sistema operativo para ser ejecutada directamente en los vehículos. Cabe destacar que ya tiene un acuerdo con Nvidia y con 28 marcas importantes del mundo de la automoción, han comenzado a implementarlo en los primeros coches a finales de 2014.

2 OBJETIVO Y ALCANCE DEL PROYECTO

Como se ha comentado antes, hoy en día existen aplicaciones móviles para cualquier utilidad, desde lo más cotidiano a lo más inverosímil. La utilidad de la aplicación móvil final será la de ayudar al conductor de un vehículo a buscar el parking más cercano y ofrecerle información acerca de dicho parking (horarios, precios, plazas libres…) Para ello, la aplicación usará la API del GPS de Android para obtener información sobre la ubicación actual del vehículo, una vez conocida su ubicación, la aplicación ofrecerá una lista de los parkings más cercanos a la ubicación del vehículo. El usuario podrá modificar los parámetros de búsqueda como la distancia o el precio, y ordenar los resultados en función de dichos parámetros.

Los principales objetivos del proyecto son:

- Estudio de su viabilidad como negocio futuro.
- Desarrollo de la aplicación móvil ParkApp.

En el estudio de la viabilidad del proyecto como negocio, se van a analizar las diferentes alternativas que existen en el mercado actual, la implementación como negocio futuro, los costes asociados a la realización del proyecto y el futuro de este como negocio con todas sus funcionalidades.

Se mostrarán las tecnologías utilizadas en el desarrollo del proyecto, los pasos seguidos en su desarrollo, se comentarán las partes fundamentales del código y se mostrarán las pruebas realizadas.

Por último, los objetivos de la versión definitiva de ParkApp destinada al usuario final serían:

- Facilitar a los usuarios la búsqueda de aparcamiento en las grandes ciudades.
- Permitir a las empresas de parking ofrecer una aplicación móvil a sus clientes.

3 ESTADO DEL ARTE

En este punto se van a detallar y analizar cuál es el estado actual de desarrollo de aplicaciones móviles similares a la aplicación final. Para ello lo primero que se va a hacer será buscar en Google Play, la tienda oficial de aplicaciones de Android, puesto que la aplicación desarrollada es para dicho sistema operativo. Donde se han encontrado tres tipos diferentes de aplicaciones que se asemejan a la aplicación final.

El primer tipo es una mezcla entre una red social y una aplicación de búsqueda de aparcamiento, ya que se basa en que la gente añada nuevos aparcamientos a su base de datos. Como ejemplo se encuentra la siguiente aplicación:

3.1 Parkopedia Estacionamiento

Figura 1: Información sobre Parkopedia Estacionamiento

El segundo tipo son aplicaciones de búsqueda de aparcamiento a nivel mundial. Como por ejemplo:

3.2 ParkMe Parking

Esta aplicación es muy completa e incluso te permite reservar plaza en algunos parking que ofrezcan dicho servicio.

Figura 2: Información sobre ParkMe Parking

3.3 Parking

Otro ejemplo más de aplicación de búsqueda de aparcamiento a nivel global.

Figura 3: Información sobre Parking

Y por último, el tercer tipo son aplicaciones de búsqueda de aparcamiento a nivel de un país o una ciudad. A continuación se muestran algunos ejemplos.

3.4 m.Parking

El primer ejemplo se trata de una aplicación de gestión de aparcamiento para una ciudad lituana.

Figura 4: Información sobre m.Parking

3.5 Parking Manijak

Este otro ejemplo permite la búsqueda de aparcamiento para los usuarios de Serbia.

Figura 5: Información sobre Parking Manijak

También han aparecido un nuevo tipo de aplicaciones de aparcamiento debido al auge que existe en muchas ciudades de la llamada zona azul. Estas aplicaciones sirven para realizar el pago del importe por estacionar el vehículo y se caracterizan por ser a nivel local. Como por ejemplo:

3.6 apparkB

Figura 6: Información sobre apparkB

4 TECNOLOGÍAS EMPLEADAS

En el desarrollo de ParkApp se ha utilizado el entorno de desarrollo y las tecnologías que se describen en este apartado.

4.1 Entorno de desarrollo

- Hardware:
  - Ordenador portátil Asus S550CB
  - Teléfono móvil OnePlus One
- Software:
  - Sistemas operativos:
    - Windows 8
    - Android 5.1.1
  - Emuladores de sistemas operativos:
    - Emulador Android integrado en Android Studio
  - Servidor:
    - XAMPP (Apache)
  - Bases de datos:
    - MySQL (servidor)
    - SQLite (bases de datos locales en el dispositivo móvil)
  - IDEs:
    - Android Studio
    - Eclipse
  - Otros:
    - Microsoft Word (redacción de la memoria)

4.2 Tecnologías empleadas

4.2.1 Android

Para el desarrollo de ParkApp se hace necesaria una aplicación móvil. Hoy en día existen dos plataformas móviles ampliamente extendidas en el mercado: Android e iOS. Se ha elegido Android por varios motivos. Para empezar, cambiar de Java a Android es relativamente sencillo: solo hay que familiarizarse con una serie de conceptos propios de Android como las *activities*.
A diferencia de las aplicaciones de programación orientada a objetos tradicionales, las *activities* son unas clases especiales que forman la estructura básica de toda aplicación Android. Asimismo, al desarrollar en Android o cualquier otra plataforma móvil hay que tener en cuenta que no se dispone de potencia ilimitada (ahorro de batería), o que el sistema Android puede decidir en cualquier momento "matar" nuestra aplicación. El reto está en asumir el compromiso entre rendimiento y ahorro de batería: Para ParkApp esto no supone un gran inconveniente, puesto que la aplicación solo realiza consultas al servidor cuando el cliente está interesado en buscar un parking cercano.
Para el desarrollo Android se tiene presente el diseño de las *activities*, y para ello, se utilizan ficheros XML. El código responde a interacciones con los elementos XML cuando se enlazan con el código correctamente.
A la hora de desarrollar en Android, la primera decisión es la elección del entorno de desarrollo.
Los dos IDE más extendidos en el desarrollo de Android son Eclipse (con el plugin de Android Development Tools) y Android Studio. Tras empezar el desarrollo de ParkApp en Eclipse, más tarde se migró a Android Studio. Las razones son la falta de estabilidad del IDE de Eclipse para Android, así como el hecho de que Google dejó de soportar las nuevas actualizaciones de Android Development Tools en Eclipse.
Como ParkApp requiere el almacenamiento local de datos que provienen del servidor, se hizo necesaria la elección de una base de datos local. Tras investigar, se vio que SQLite es la base de datos más extendida para Android, y la interacción Android-SQLite resulta sencilla si se utilizan una serie de clases y métodos ya escritos y disponibles para el público.

4.2.2 Servidor central y bases de datos del servidor

Se ha elegido Apache como servidor web que responda a las peticiones de los clientes. La base de datos del servidor se desarrollará en el lenguaje MySQL.

4.2.2.1 Apache

Desarrollado en el marco del proyecto de HTTP Server de Apache Software Foundation, Apache es el servidor HTTP más utilizado en la actualidad, y tiene una serie de ventajas con respecto a sus competidores principales como pueden ser el hecho de que Apache Server es altamente configurable o que proporciona bases de datos de autenticación y con negociación de contenido.
Sin embargo, la mayor arma de Apache, y que ha permitido su total extensión en el ámbito de servidores HTTP, es su carácter abierto, multi-plataforma y extensible. Además, al estar tan extendido su uso, es también muy fácil encontrar ayuda y tutoriales sobre *best practices* en Internet.
Es muy usado hoy en día en conjunción a MySQL y PHP en el desarrollo de páginas dinámicas en Internet. Una de las grandes ventajas que proporciona el uso de Apache es además su integración en paquetes junto a MySQL, para el desarrollo de webs o bases de datos dinámicas desde un servidor local. En el desarrollo de ParkApp, se ha utilizado el paquete XAMPP. Esto ha permitido realizar pruebas sin la necesidad de subir la base de datos a la red, lo cual ha resultado en evidente ahorro de tiempo (esto ha permitido previsualizar el resultado final, y llevar a cabo eficientes test locales).

4.2.2.2 MySQL

Para la gestión de bases de datos en el desarrollo de ParkApp, MySQL ha sido el sistema preferido. MySQL es un sistema de gestión de base de datos relacional, multiusuario y con más de seis millones de instalaciones. Además, está desarrollado como software libre, aunque siguiendo un esquema de licencia dual. Está ofrecido bajo licencia GNU GPL para usos compatibles con ésta. Por otro lado, empresas que quieran incorporar MySQL en productos privativos deben una licencia que permite dicho uso.
La distribución libre de MySQL está mantenida por la amplísima comunidad de programadores que da soporte a código Open Source.
Como ventajas de MySQL, cabe destacar que es una tecnología que está adaptada para funcionar en equipos de múltiples procesadores, proporcionando una muy alta fidelidad y velocidad de respuesta. Las características que se han ido añadiendo a MySQL han tenido siempre en cuenta que el principal objetivo de MySQL es garantizar que estas no comprometen la estabilidad o el rendimiento del sistema.
En ParkApp se ha utilizado XAMPP en las pruebas con la base de datos MYSQL. XAMPP nos permite gestionar bases de datos y crear distintas tablas, así como visualizar las filas de la base de datos creada, ejecutar queries de prueba y ver el tiempo de ejecución de las mismas.
Sin conocimiento previo de bases de datos, la curva de aprendizaje SQL ha resultado muy corta gracias en parte a las facilidades que ha dado XAMPP.

4.3 Alternativas existentes a las tecnologías empleadas

En algunos casos la elección de una determinada tecnología se ha debido a que ya se disponía de experiencia en dicho campo, mientras que en otros ha sido el resultado de investigación sobre alternativas y del análisis de ventajas e inconvenientes de cada una. A la hora de elegir cada tecnología en el desarrollo de ParkApp se han tenido en cuenta los siguientes factores:

- Conocimiento previo
- Curva de aprendizaje
- Extensión y popularidad (existencia de tutoriales, foros de apoyo)
- Perspectiva de desarrollo futuro
- Precio

A continuación se exponen algunas de las muchas alternativas que hay en cada a cada una de las tecnologías escogidas. Se han intentado recoger las principales, y se explicarán los motivos por los cuales se descartó su uso.

4.3.1 Alternativas a Android

Según la IDC, la venta de *smartphones* ha continuado creciendo un 13% año tras año y en el segundo trimestre de 2015 Android sigue dominando el mercado a nivel global con un 82.8%, muy por encima del 13.9% que tiene iOS, su principal competidor.

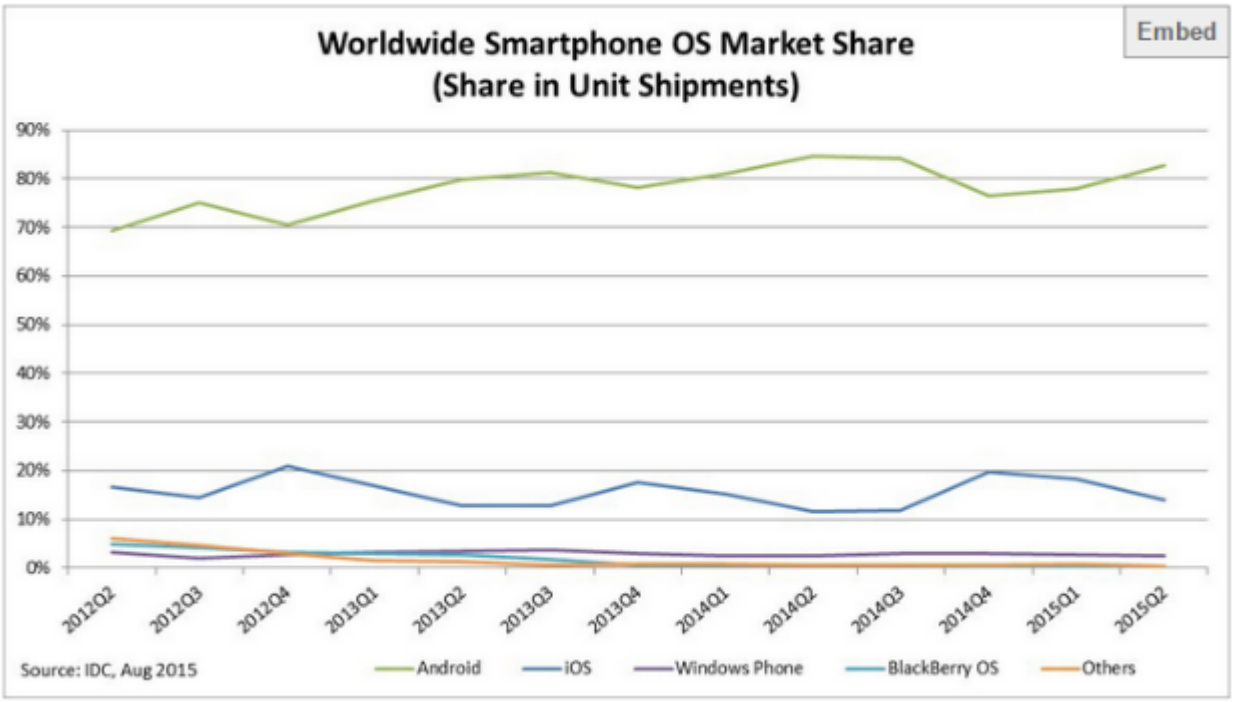

Figura 7: Gráfica sobre la venta de *smartphones* según su sistema operativo

4.3.1.1 Apple iOS

Apple vende alrededor del 15% de móviles del mercado, lo cual es casi un 70% por debajo de su rival en el mercado de móviles, Android. Sin embargo, iOS tiene una comunidad enorme de desarrolladores, y el mercado App Store mueve incluso más dinero que Google Play (a finales de 2013, el mercado Apple hacía más dinero que toda la competencia junta). Esto es debido a los distintos perfiles de los usuarios de Apple y Android. Los usuarios de Apple están por lo general más dispuestos a gastar dinero en aplicaciones.
Sin embargo, ParkApp está pensada para ser gratuita, porque será usada por los usuarios que busquen parkings cercanos o por los clientes de los parkings que den apoyo a la aplicación en el futuro. Por ello, el hecho de que Android ocupa actualmente más del 80% de las ventas de dispositivos móviles ha sido definitivo a la hora de elegir dicha plataforma.
Además, el desarrollo de iOS es costoso, ya que precisa de ordenador Apple y de pago de cuota anual de 99 dólares, y a la hora del desarrollo de ParkApp uno de los objetivos fijados ha sido el que éste sea lo más económico posible.
Sin embargo, no es recomendable despreciar un 15% de mercado potencial a la hora de desarrollar un producto, así que en futuras versiones de ParkApp sí se tendría en cuenta el desarrollo de versiones iOS.

4.3.1.2 Windows Phone

El ambicioso proyecto Windows Phone ha resultado en un potente sistema operativo, que solo en el primer trimestre de 2015 vendió 9.2 millones de unidades y casi un 3% del mercado. Sin embargo, esto se antoja insuficiente a la hora de elegir la plataforma de desarrollo en un proyecto que busca llegar al mayor número de personas posible.

4.3.2 Alternativas a Apache

Existen muchas alternativas al servidor Apache. Sin embargo, se ha elegido Apache por las ventajas ya comentadas anteriormente. Se van a mencionar algunas de las alternativas que se han considerado:

4.3.2.1 Thttpd

Distribuido en software libre y disponible para plataformas UNIX, tiene como mejores prestaciones su simplicidad, poco tamaño, portabilidad, rapidez y seguridad, empleando los requerimientos mínimos de un servidor HTTP. Sin embargo, tiene una serie de inconvenientes, como por ejemplo que no soporta https, esto hizo que se descartara como alternativa.

4.3.2.2 Nginx

Servidor web que ha crecido mucho en los últimos años, es recomendado para sitios web estáticos que necesiten alta velocidad, con poco uso de la memoria y sitios web que funcionan con un servidor dedicado. Al estar basado en eventos y no en procesos (como Apache, en el que cada conexión requiere un hilo, lo que aumenta la carga considerablemente), nginx consume mucho menos memoria. Además, cuando hay muchas conexiones simultáneas, nginx ofrece muy altas velocidades de transferencia.

No se eligió Nginx por el hecho de tener Apache una mayor comunidad de desarrolladores. Sin embargo, podría haber sido la alternativa más eficiente a la hora de procesar las conexiones de clientes en ParkApp.

4.3.3 Alternativas a MySQL

La principal alternativa a MySQL es la siguiente:

4.3.3.1 MariaDB

Es considerada la evolución de MySQL, y son compatibles. De hecho, en la migración MySQL-MariaDB no es necesario cambiar modificar el código. MariaDB es un sistema de gestión de bases de datos derivado de MySQL con licencia totalmente gratuita GPL. Es desarrollado por Michael Widenius (fundador de MySQL) y la comunidad de desarrolladores de software libre. Se considera que existen una serie de ventajas de MariaDB con respecto a MySQL:

- Motores de almacenamiento nuevos, como Aria, que reemplaza MyISAM.
- Mejoras de velocidad, sobre todo cuando se utiliza el motor de almacenamiento Aria. El motivo es que Aria cachea datos en tablas temporales de memoria, mejorando así el rendimiento de MyISAM, que utilizaba el disco duro.
- Nuevas tablas de sistema que almacenan estadísticas que pueden ser útiles para realizar la optimización de la base de datos.

MariaDB no supone un salto abrupto desde MySQL, por lo que incluso se podría hacer el trasvase de las bases de datos de ParkApp a MariaDB sin problemas. MySQL fue la base de datos elegida por encontrarse en el paquete de desarrollo XAMPP, aunque en versiones futuras se tendrá en cuenta MariaDB como evolución natural de MySQL.

5 DESARROLLO DE PARKAPP

5.1 Introducción

ParkApp es una aplicación para la búsqueda de parkings cercanos. Para ello el usuario solo tiene que descargar la aplicación de forma gratuita e instalarla en su dispositivo móvil. Una vez instalada la aplicación, el usuario verá su ubicación en el mapa y deberá introducir la distancia deseada. Entonces la aplicación buscará en una base de datos todos los parkings que estén incluido en el radio deseado. Cuando haya finalizado la búsqueda, mostrará los resultados en una lista personalizada. También se ha desarrollado un menú en ParkApp desde el que se puede modificar la vista del mapa, acceder a una pantalla en la que cambiar algunos parámetros, tales como, el idioma o la unidad de longitud, acceder a una pantalla Acerca de y salir de la aplicación. La capacidad de mejora de ParkApp es enorme pudiendo incluir rutas desde la ubicación actual hasta el parking deseado o incluso hacer uso de la herramienta Navigation de Google que te guía hasta dicho parking.

5.2 Esquema básico de funcionamiento del sistema

Cada uno de los usuarios podrá realizar peticiones al servidor de la base de datos mediante la aplicación ParkApp instalada en su dispositivo Android.

Figura 8: Esquema básico funcionamiento del sistema

5.3 Diagramas de casos de uso

Los diagramas de casos de uso permiten diferenciar los actores que interactúan con el sistema, las relaciones entre ellos y las acciones que puede realizar cada uno dentro del sistema.
Este tipo de diagramas es comprensible tanto por clientes como por usuarios, y representa los requisitos funcionales del sistema, utilizándose como base para un desarrollo iterativo e incremental. Los diagramas de casos de uso tienen tres elementos:

- Actores: Son los usuarios del sistema. Un actor puede ser una persona, un conjunto de personas, un sistema hardware o un sistema software. Los actores representan un rol, que puede desempeñar alguien que necesita intercambiar información con el sistema.
- Casos de uso: Un caso de uso describe una forma concreta de utilizar parte de la funcionalidad de un sistema. La colección de todos los casos de uso describe toda la funcionalidad del sistema.
- Comunicación entre actores y casos de uso: Cada actor ejecuta un número específico de casos de uso en la aplicación. Por eso se dice que hay comunicación entre actores y casos de uso.

5.3.1 Actores

Figura 9: Actores del sistema

5.3.2 Diagramas de casos de uso del usuario

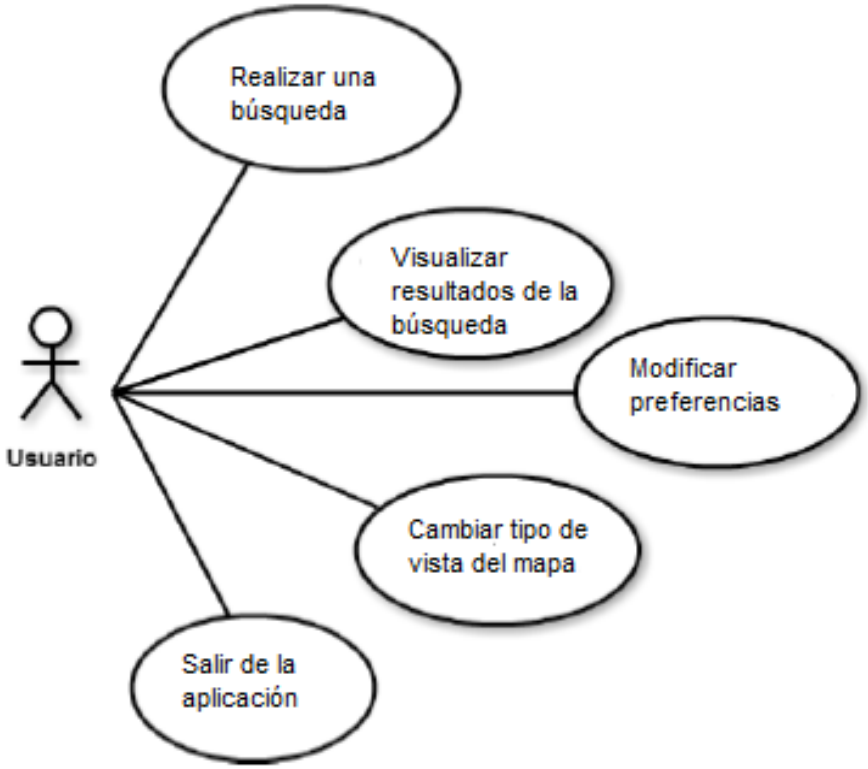

Figura 10: Diagrama de casos de uso del usuario

5.4 Diagramas de clases

Se van a evaluar las distintas clases que componen la aplicación ParkApp.

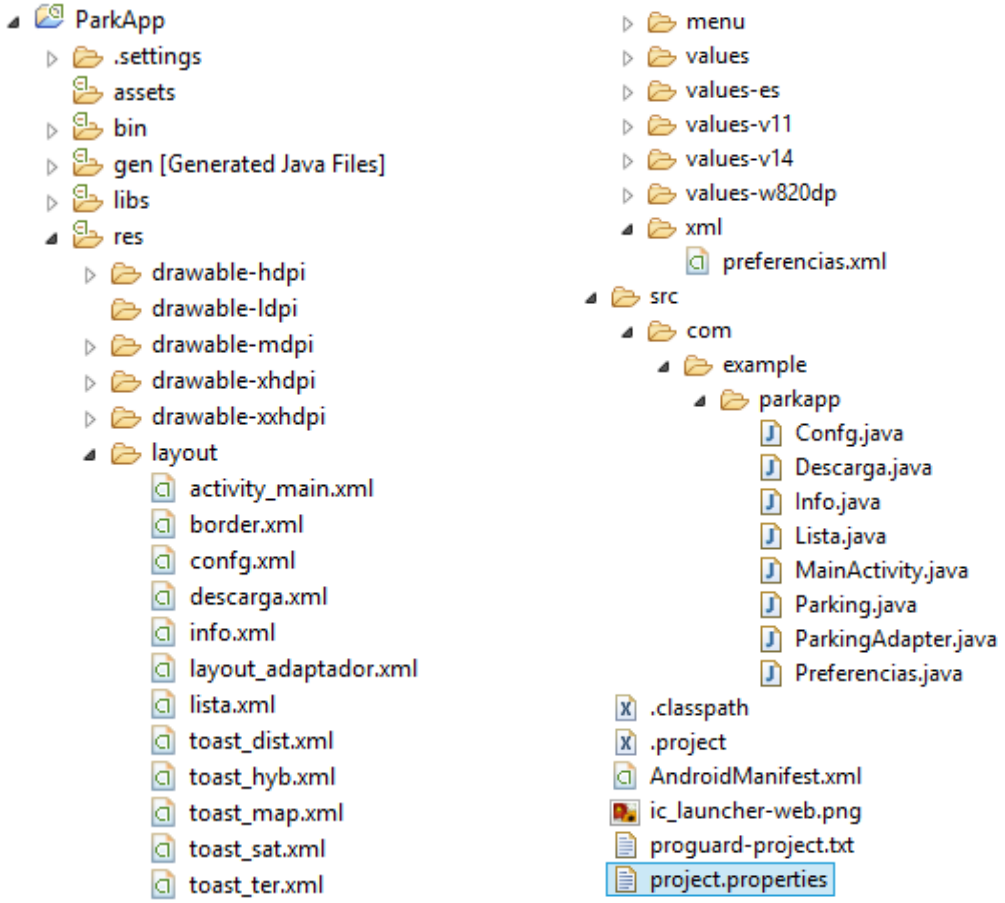

Figura 11: Clases que componen ParkApp

Para comenzar, mencionar que en una aplicación Android hay código Java (en el caso de ParkApp en la carpeta com.example.parkapp), y también hay código XML que se referencia desde el código Java. Estos archivos XML se encuentran en la carpeta res (resources), y son imprescindibles para el correcto desarrollo de la aplicación. En la carpeta layout hay varios archivos XML, uno por cada elemento gráfico de la aplicación, mientras que en la carpeta drawable se colocan las imágenes que luego se referencian en el resto del código. También es importante destacar las carpetas values y values-es puesto que en ellas se guardarán los valores que tomarán las cadenas según el idioma que se elija en la aplicación, inglés o español respectivamente. Por último, destacar el archivo AndroidManifest.xml puesto que en dicho archivo se configuran las licencias y los permisos que tendrá la aplicación, como por ejemplo, la licencia para usar los mapas de Google.

5.4.1 Clase MainActivity

Carga una interfaz de usuario que consta de un campo de texto donde el usuario inserta la distancia deseada, un botón para iniciar la búsqueda, un menú que se explicará más detalladamente y un mapa de Google cuya principal característica es que se trata de un *fragment*.

Es utilizado en esta clase porque los mapas de Google heredan de la clase *fragment* y el elemento principal de la clase principal de ParkApp es el mapa donde el usuario verá su posición marcada.

Otra parte importante de esta clase es el menú de la aplicación (*Action bar menu*), este menú en

Android fue añadido a partir de Android 3.0 y su función era la de eliminar los botones físicos en los dispositivos Android. Este menú es un desplegable desde el cual el usuario podrá acceder a distintas funcionalidades de la aplicación.

Figura 12: Captura de la pantalla principal con el menú desplegado

Para analizar el menú de ParkApp se hará directamente sobre el código:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater infl = getMenuInflater();
    infl.inflate(R.menu.main, menu);

    return super.onCreateOptionsMenu(menu);
}
```

Figura 13: Fragmento de código del *inflater* del menú

Lo primero que hay que hacer es crear un *inflater* que se encargará de inflar el *layout* que contendrá el menú.

El siguiente método muestra las distintas opciones que mostrará el menú de ParkApp:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_sat:
            map.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
            textIconToast(R.layout.toast_sat, Toast.LENGTH_LONG);
            break;
        case R.id.menu_map:
            map.setMapType(GoogleMap.MAP_TYPE_NORMAL);
            textIconToast(R.layout.toast_map, Toast.LENGTH_LONG);
            break;
        case R.id.menu_hyb:
            map.setMapType(GoogleMap.MAP_TYPE_HYBRID);
            textIconToast(R.layout.toast_hyb, Toast.LENGTH_LONG);
            break;
        case R.id.menu_ter:
            map.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
            textIconToast(R.layout.toast_ter, Toast.LENGTH_LONG);
            break;
        case R.id.menu_confg:
            Intent c = new Intent(getApplicationContext(), Confg.class);
            startActivity(c);
            break;
        case R.id.menu_info:
            Intent i = new Intent(getApplicationContext(), Info.class);
            startActivity(i);
            break;
        case R.id.menu_salir:
            finish();
            break;
    }
    return super.onOptionsItemSelected(item);
}
```

Figura 14: Fragmento de código del menú

Los cuatro primeros casos muestran las distintas vistas del mapa que puede elegir el usuario (satélite, normal, híbrido y terreno), como se ve en la imagen cada una de estas opciones muestra un *toast* en pantalla al ser seleccionada.

Los siguientes casos lanzan un *intent* que llevará al usuario a la pantalla de configuración o a la de información respectivamente.

Por último, existe la opción de salir de la aplicación directamente sin pulsar el botón de *home* o ningún otro botón físico.

5.4.2 Clase Descarga

Esta clase es llamada cada vez que el usuario pulsa el botón para realizar una búsqueda, y su función es bastante simple, generar una animación mientras la aplicación se conectase con el servidor. Una vez el servidor enviase una respuesta y esta sea procesada, se lanzaría la clase Lista.

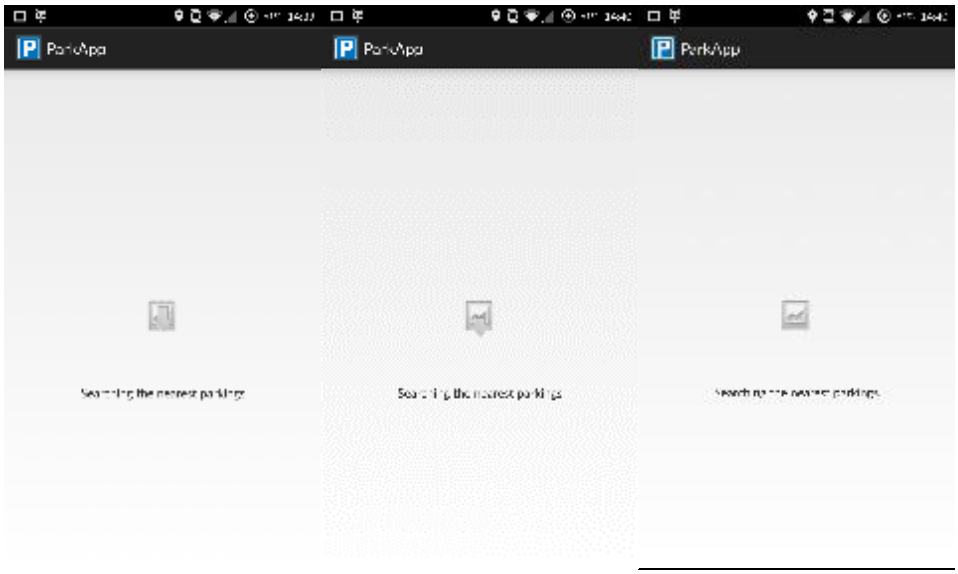

Figura 15: Capturas de la animación

5.4.3 Clase Lista

Es la clase encarga de generar la lista personalizada donde se mostrarán los distintos parkings que cumplen con los requisitos de la búsqueda. Lo primero que se realiza es crear un *array* de tipo Parking donde cada elemento tendrá los campos que se muestran a continuación:

```
public class Parking {
    private String nombre, distancia, plazas, horario, precio;

    public Parking(String nombre, String distancia, String plazas, String horario, String precio) {
        this.nombre = nombre;
        this.distancia = distancia;
        this.plazas = plazas;
        this.horario = horario;
        this.precio = precio;
    }
```

Figura 16: Fragmento de código del tipo Parking

Dicho *array* se rellenará con los campos de los parkings que cumplan las condiciones de la búsqueda. Una vez rellenado se inflará el adaptador que mostrará cada uno de los elementos en la posición y con el tamaño deseado dentro de la lista como se muestra en la siguiente figura.

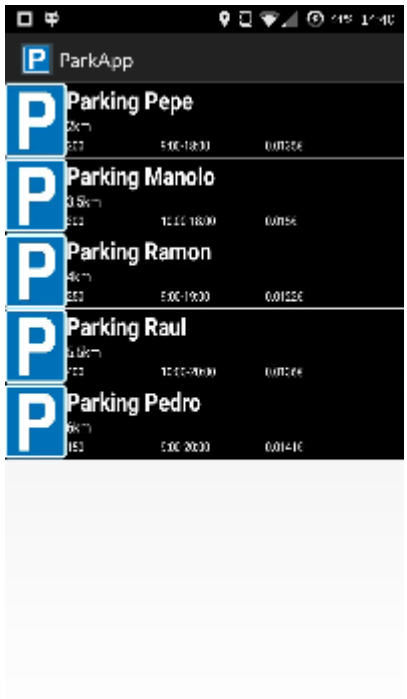

Figura 17: Captura de pantalla de la lista

5.4.4 Clase Confg y Preferencias

Estás clases serán tratadas conjuntamente puesto que la clase Confg actúa como un mero trámite ya que las *preferences* en Android se tratan como un *fragment* y no como una *activity* en sí. La clase Preferencias cuenta con un *listener* que se mantiene a la escucha hasta que se produce un cambio en el idioma de la aplicación o en la unidad de la distancia.

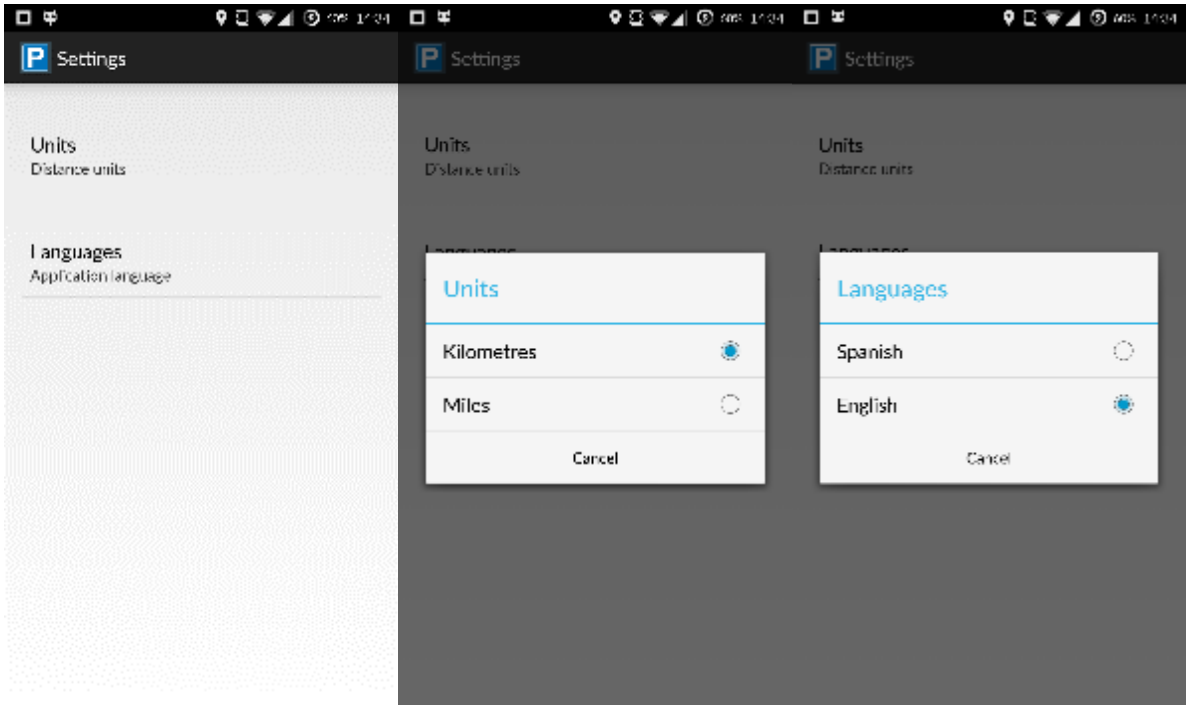

Figura 18: Capturas de la pantalla de configuración

Si se produce algún cambio en estos parámetros se volverá a la pantalla principal de la aplicación e incluso se puede mostrar un pequeño mensaje en pantalla como muestra el código.

```java
public class Preferencias extends PreferenceFragment implements OnSharedPreferenceChangeListener {
    @Override
    public void onCreate(Bundle savedInstanceState){
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.preferencias);

        this.getPreferenceScreen().getSharedPreferences().registerOnSharedPreferenceChangeListener(this);
    }

    @Override
    public void onSharedPreferenceChanged (SharedPreferences sharedPreferences, String key) {
        if (key.equals("idioma")) {
            Intent i = new Intent(getActivity(), MainActivity.class);
            i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(i);
        }
        if (key.equals("unidad")) {
            Toast.makeText(getActivity(), R.string.toast_unidad, Toast.LENGTH_LONG).show();
            Intent i = new Intent(getActivity(), MainActivity.class);
            i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(i);
        }
    }
}
```

Figura 19: Fragmento de código de la clase Preferencias

5.4.5 Clase Info

Por último, la aplicación cuenta con la clase Info que se trata de un simple *layout* donde se muestra información básica sobre la aplicación, su versión y el desarrollador.

Figura 20: Captura de la pantalla de información

5.5 Diseños de bases locales

ParkApp está diseñada para trabajar con una base de datos MySQL ubicada en un servidor central a la que la aplicación realizará peticiones mediante JSON. En estas versiones preliminares se ha trabajó al inicio con una base local SQLite creada mediante otra aplicación diseñada para tal uso, denominada Base de Datos y cuyo código se muestra a continuación:

```java
public class VentanaInsercion extends ActionBarActivity {

    private EditText txt_nombre, txt_lat, txt_long, txt_plazas, txt_horario, txt_precio;
    private Button btn_guardar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_ventana_insercion);

        txt_nombre = (EditText)findViewById(R.id.edit_nombre);
        txt_lat = (EditText) findViewById(R.id.edit_lat);
        txt_long = (EditText) findViewById(R.id.edit_long);
        txt_plazas = (EditText) findViewById(R.id.edit_plazas);
        txt_horario = (EditText) findViewById(R.id.edit_horario);
        txt_precio = (EditText) findViewById(R.id.edit_precio);
        btn_guardar = (Button) findViewById(R.id.bttn_guardar);

        btn_guardar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                String nombre = txt_nombre.getText().toString();
                double lat = Double.parseDouble(txt_lat.getText().toString());
                double lon = Double.parseDouble(txt_long.getText().toString());
                String plazas = txt_plazas.getText().toString();
                String horario = txt_horario.getText().toString();
                String precio = txt_precio.getText().toString();

                Conexion cn = new Conexion(getApplicationContext(), "DBusers.db", null, 1);
                SQLiteDatabase db = cn.getWritableDatabase();
                cn.InsertarUsuario(db, nombre, lat, lon, plazas, horario, precio);
            }
        });
    }
}
```

Figura 21: Fragmento código aplicación Base de Datos

Esta sencilla aplicación crea una base de datos SQLite en el teléfono y crea un nuevo registro en dicha base de datos cada vez que el usuario rellena los campos y pulsa el botón guardar.

Figura 22: Captura de la pantalla de inserción

Más adelante, se hizo uso de la herramienta XAMPP y se realizaron consultas sobre una base de datos MySQL en un servidor Apache. Esta funcionalidad será implementada en las versiones definitivas de ParkApp cuando ésta trabaje con un servidor real que contenga los distintos parkings.

5.6 Evaluación y pruebas

Se diseñan una serie de pruebas para comprobar el correcto funcionamiento de la aplicación ParkApp. Las pruebas van a consistir en comprobar la correcta ejecución de cada una de las funciones de ParkApp y de las diversas órdenes que podría realizar un usuario. Cabe mencionar que los distintos parkings que aparecen en dichas pruebas son ficticios y han sido creados con la intención de probar la aplicación.

Número	Prueba	Resultado
1	Crear marcador en el mapa.	OK
2	Dibujar círculo con la distancia.	OK
3	Comprobar distancia entre marcadores.	OK
4	Cambiar vista del mapa.	OK
5	Mostrar mensaje si se introduce una distancia errónea.	OK
6	Cambiar parámetros correctamente.	OK
7	Mostrar lista de ejemplo.	OK

Tabla 1: Listado de pruebas

5.6.1 Prueba 1: Crear marcador en el mapa

La prueba consiste en mostrar un marcador en el mapa de la pantalla principal, puesto que la aplicación mostrará la ubicación actual del usuario. Para ello se hace uso de la siguiente instrucción:

```
final MarkerOptions marcador = new MarkerOptions();
marcador.position(posicion);
marcador.title("R.string.ubicacion");
map.addMarker(marcador);
```

Figura 23: Fragmento código marcador

Se comprueba en la siguiente imagen que la prueba se ha realizado correctamente:

Figura 24: Captura de la pantalla principal con el marcador

5.6.2 Prueba 2: Dibujar círculo con la distancia

La prueba consiste en mostrar en el mapa de la pantalla principal un círculo alrededor del marcador con la ubicación del usuario cuyo radio es la distancia introducida por dicho usuario. Para ello se hace uso de la siguiente instrucción:

```
circulo = map.addCircle(new CircleOptions()
 .center(posicion)
 .radius(dist)
 .strokeColor(Color.BLACK)
 .fillColor(Color.TRANSPARENT));
```

Figura 25: Fragmento código círculo

Se comprueba en la siguiente imagen que la prueba se ha realizado correctamente:

Figura 26: Captura de la pantalla principal con el c´rculo

5.6.3 Prueba 3: Comprobar distancia entre marcadores

La prueba consiste en mostrar dos marcadores en el mapa, un círculo cuyo radio es la distancia introducida por el usuario y un toast que muestra un mensaje distinto si el segundo marcador se encuentra dentro del círculo o no.

Esta prueba se realiza porque con este método se comprobará si cada uno de los parkings de la base de datos debe ser mostrado en la lista o no. Para llevar a cabo esta prueba se hace uso del siguiente bloque de instrucciones:

```
Location.distanceBetween( marcador.getPosition().latitude, marcador.getPosition().longitude,
        prueba.getPosition().latitude, prueba.getPosition().longitude, distance);

    if(distance[0] > dist){
        Toast.makeText(getBaseContext(), R.string.toast_fuera, Toast.LENGTH_LONG).show();
    } else {
        Toast.makeText(getBaseContext(), R.string.toast_dentro, Toast.LENGTH_LONG).show();
    }
```

Figura 27: Fragmento código distncia entre marcadores

Se comprueba en las siguientes imágenes que la prueba se ha realizado correctamente:

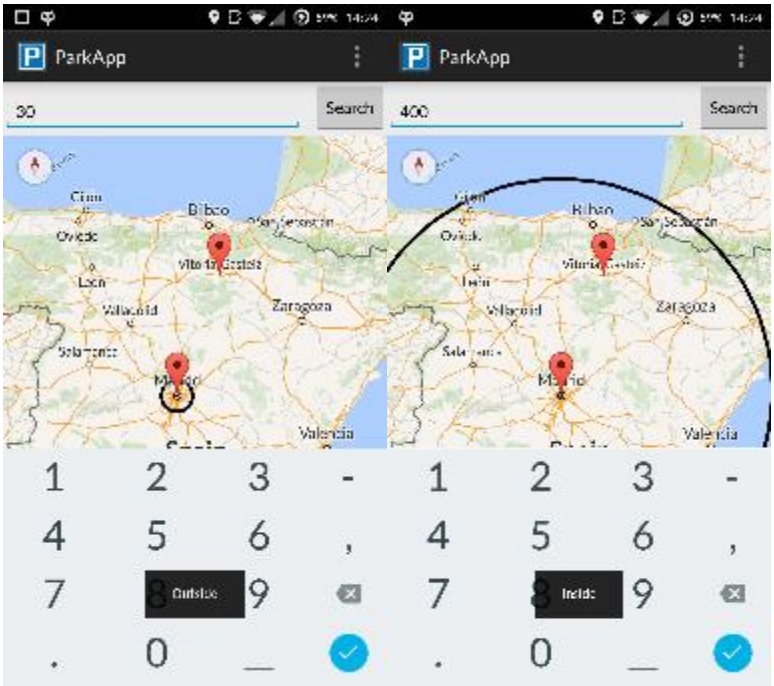

Figura 28: Capturas de pantalla distancia entre marcadores

5.6.4 Prueba 4: Cambiar vista del mapa

La prueba consiste en mostrar algunas de las vistas que puede tener el mapa principal y con ello comprobar que se puede alternar entre las distintas vistas del mapa correctamente. Para ello basta con pulsar en la opción correspondiente en el menú de la aplicación.

Se comprueba en las siguientes imágenes que la prueba se ha realizado correctamente:

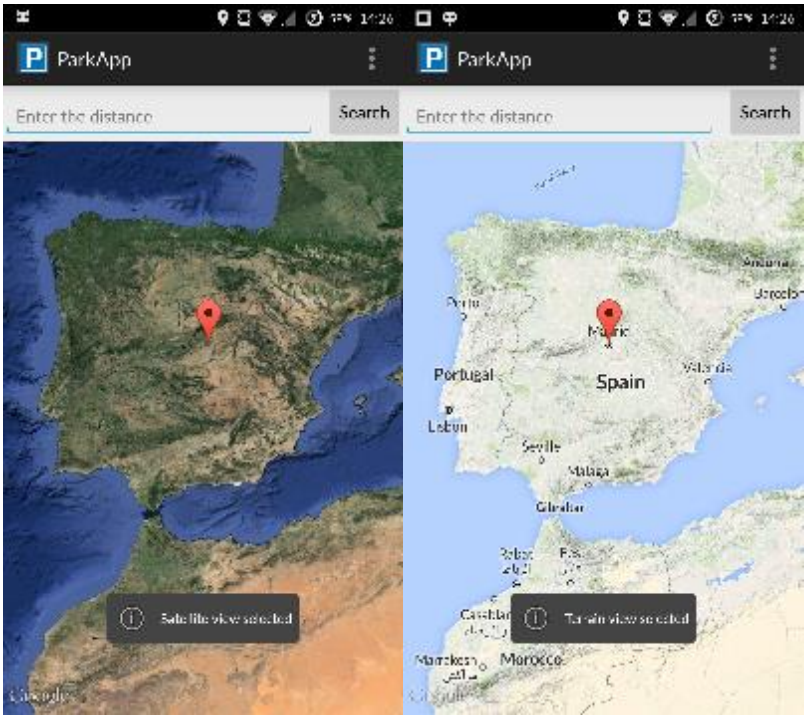

Figura 29: Capturas de pantalla vistas del mapa

5.6.5 Prueba 5: Mostrar mensaje si se introduce una distancia errónea

La prueba consiste en comprobar que aparece un toast con un mensaje de advertencia para el usuario para que introduzca un valor de la distancia adecuado. Este mensaje aparece automáticamente si el usuario pulsa el botón de búsqueda y el valor introducido es incorrecto.

Se comprueba en la siguiente imagen que la prueba se ha realizado correctamente:

5.6.6 Prueba 6: Cambiar parámetros correctamente

La prueba consiste en comprobar que al cambiar el valor de alguno de los parámetros en la clase Preferencias, estos cambios se reflejan en la aplicación. Por el hecho de ser más visual se realizará la prueba modificando el idioma de la aplicación. Para ello basta con pulsar en el parámetro Idioma dentro de la clase Preferencias y pulsar sobre el idioma que no está seleccionado actualmente.

Se comprueba en las siguientes imágenes que la prueba se ha realizado correctamente:

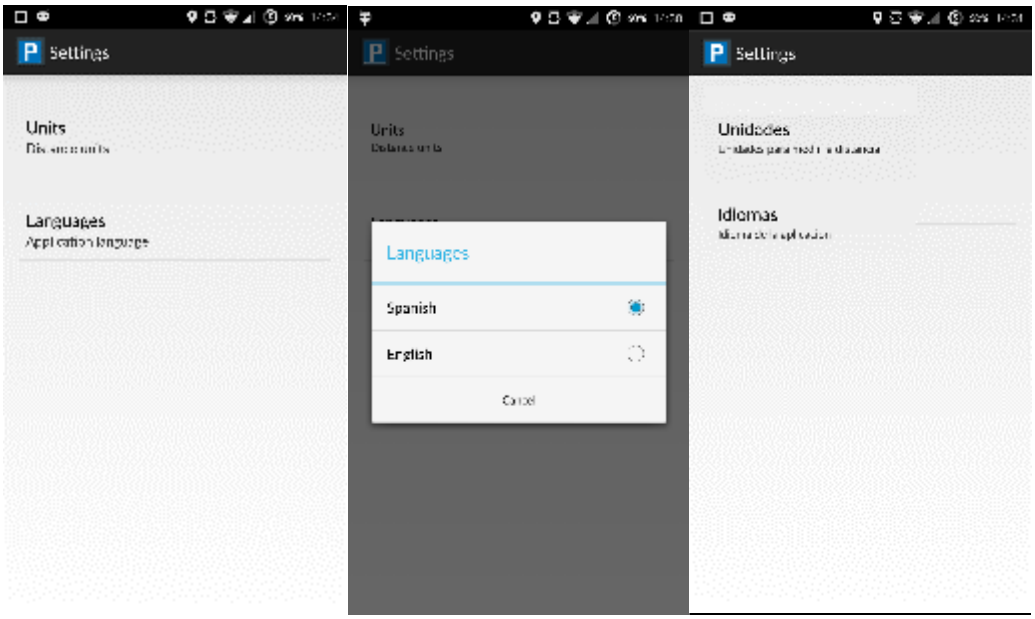

Figura 30: Capturas de pantalla cambio de idioma

5.6.7 Prueba 7: Mostrar lista de ejemplo

La prueba consiste en pulsar el botón de búsqueda para que la aplicación muestre una lista con los parkings. Esta prueba se realiza para comprobar que cada uno de los elementos de la lista aparece correctamente y de la forma deseada.

Se comprueba en la siguiente imagen que la prueba se ha realizado correctamente:

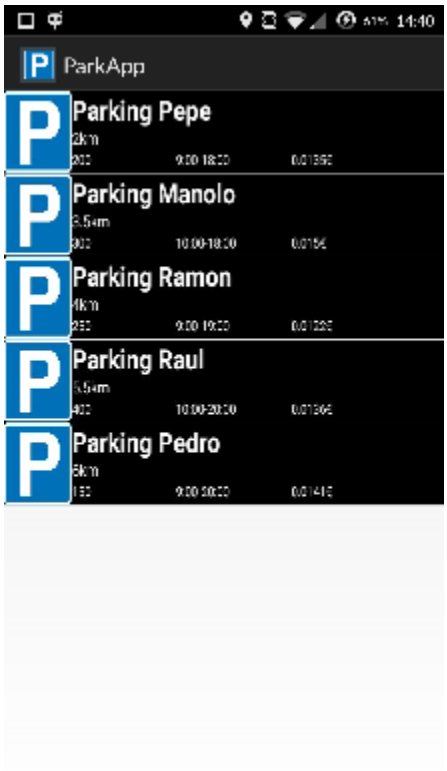

Figura 31: Captura de pantalla de la lista

6 PLAN DE IMPLEMENTACIÓN DE PARKAPP

6.1 Concepto de negocio

Esta aplicación llamada ParkApp ofrece a las empresas propietarias de los parkings las siguientes funcionalidades principales:

- Gestión de parkings
- Actualización de precio, horario, etc… de manera inmediata
- Herramienta de búsqueda de parking para sus clientes

Todas estas funcionalidades serán posibles mediante el servidor central al que se conectará la aplicación y mediante la aplicación móvil en sí. También se tienen pensadas añadir funcionalidades extra como la de guardar parkings como favoritos o que la aplicación te guíe hasta el parking deseado desde la ubicación actual.

Como se comentó al inicio, también se pretende que la aplicación esté disponible tanto para Android como para iOS, por lo que los futuros usuarios podrán descargarlas desde las respectivas tiendas de aplicaciones (Google Play, App Store).

A la hora de realizar el modelo de negocio, la primera intención es centrarse en parkings independientes en España, con el objetivo de un mantenimiento sencillo y poder profundizar en el mercado. Estos pequeños parkings también tendrán la intención y la posibilidad de crecer junto con la aplicación.

Después de realizar un estudio de mercado, cabe destacar que según la ASESGA existen 1.709 aparcamientos, que gestionan 906.001 plazas, distribuidos entre las más de 1.000 empresas afiliadas. Por lo que el primer objetivo es conseguir en el primer año tener 10 empresas asociadas a ParkApp, lo que supondría un 1% del mercado. Una vez se alcance esta cifra, el objetivo sería continuar creciendo hasta alcanzar 50 empresas asociadas en España en un plazo de 3-4 años, lo que supondría un 5% del mercado, una cifra bastante alta para tratarse de una aplicación emergente.

El modelo de negocio se basa en suscripciones al programa ParkApp por parte de los aparcamientos, en las que se ofrecerá un periodo de prueba gratuito de 2 meses tras el cual la empresa deberá pagar una cuota de abonado de 20€ mensuales.

Para conseguir un producto mínimo viable, se requiere de una inversión inicial de 5.000€ para cubrir los gastos iniciales como:

- Desarrollo de las distintas versiones de ParkApp
- Desarrollo de una página web
- Mantenimiento del servidor

- Plan de marketing y comercialización
- Otros

Cabe destacar que esta cuota inicial no es un valor desorbitado puesto que si se logra el crecimiento deseado en los primeros años, se ingresaría un mínimo de 1000€ mensuales.

6.2 Presentación de la empresa

La idea de ParkApp nace del problema diario que existe en las grandes ciudades a la hora de buscar aparcamiento. En la mayoría de los casos los conductores acaban accediendo a un parking por el hecho de no poder encontrar ningún aparcamiento en la calle, sin saber cuánto tendrán que pagar al salir puesto que es lo único que les queda. También es habitual en los conductores que desconozcan la ubicación de muchos parkings, ya no solo en otras ciudades, sino incluso en su lugar de residencia.

Con el objetivo de solucionar estos problemas y utilizando las aplicaciones móviles, ParkApp ofrece una aplicación que se encarga de encontrar para el usuario todos los parkings cercanos a su ubicación, por lo que el usuario podrá decidir cuál es el parking mejor situado o el más económico con el simple hecho de mirar su móvil.

¿Por qué se usa una aplicación móvil? Como se comentó en la introducción de la memoria, el desarrollo de aplicaciones móviles se ha convertido en un mercado emergente y en el que aún queda mucho por explotar. Su evolución está siendo tremenda desde que empezaron a usarse los *smartphones* como una herramienta indispensable en el día a día.

ParkApp tiene un hueco en el mercado ya que, aunque existen distintas aplicaciones para buscar aparcamiento, está diseñada para trabajar directamente con las empresas del sector y aunar los parkings a nivel nacional. Cabe destacar, que una vez desarrollada ParkApp en unas de sus versiones finales y estar asentada en el mercado español, se podría extrapolar a distintos países.

6.3 Visión, misión y objetivos

La visión es implantar ParkApp en el mercado español mediante la colaboración con el mayor número de empresas que forman ASESGA y convertirla en una referencia para el sector y para los usuarios. Para llevar a cabo todo esto, es necesario primero publicar una versión con el producto mínimo viable, y luego ir mejorándola y añadiéndole nuevas características gradualmente.

Como se ha mencionado anteriormente, una vez desarrollada una versión final de ParkApp y haber penetrado en el mercado español, una visión mucho más ambiciosa sería la de expandirla a otros países.

La misión principal de ParkApp sería la de ofrecer a los usuarios una interfaz sencilla e intuitiva, por lo que en la pantalla principal solo se muestra un campo donde introducir la distancia y un mapa donde ver su ubicación, consiguiendo así una mayor comodidad para estos. También pretende ser una herramienta útil para las empresas que gestionan los parkings y para que un mayor público conozca sus servicios.

Por último, los principales objetivos como empresa serían:

- Empezar a establecerse en los parkings de Sevilla y sus alrededores.
- Ir consiguiendo una mayor cuota de mercado, no solo a nivel provincial.
- Llegar a convertirse en una aplicación de referencia a nivel nacional.

7 COSTE DEL PROYECTO

En este apartado se desglosan los costes involucrados en la realización del proyecto ParkApp. Los costes se van a desglosar en costes de recursos humanos y costes adicionales, para posteriormente realizar un cálculo total del coste del proyecto.

7.1 Coste de recursos humanos

El proyecto se ha realizado con los recursos de un ingeniero a una tarifa horaria de 25 euros la hora. Además, se considera que las tareas de programación han sido realizadas por un programador a una tarifa de 15 euros la hora.
Se ha llevado a cabo entre el final de Mayo y final de Septiembre de 2015 y se calcula que en total se han dedicado 435 horas de trabajo a su ejecución. A continuación, se hace un desglose de los costes en recursos humanos del proyecto ParkApp.

Concepto	Horas	Ejecutor	Coste	Acumulado
Brainstorming	10	Ingeniero	250€	250€
Planificación proyecto	15	Ingeniero	375€	625€
Estudio alternativas mercado	20	Ingeniero	500€	1.125€
Decisión funcionalidades	20	Ingeniero	500€	1.625€
Tutorial Android/SQL	30	Programador	450€	2.075€
Aplicación Android	240	Programador	3.600€	5.675€
Pruebas Servidor/DB	20	Programador	300€	5.975€
Pruebas Aplicación	20	Ingeniero	500€	6.475€
Redacción memoria	60	Ingeniero	1.500€	7.975€

Tabla 2: Coste de recursos humanos

A continuación se van a mostrar las etapas de desarrollo del proyecto a lo largo del tiempo en un diagrama de Gantt. Varias tareas se realizaron paralelamente, al estar unas relacionadas con las otras (se ha intentado siempre dar un enfoque práctico a lo aprendido, y así, por ejemplo, cuando se ha adquirido una nueva habilidad nueva al hacer un tutorial se ha ido probando ésta en el proyecto).

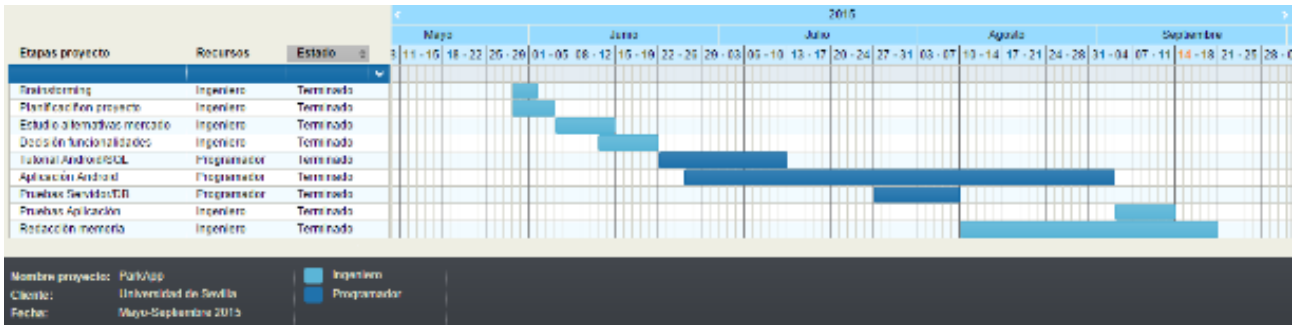

Figura 32: Diagrama de Gantt

7.2 Costes adicionales

Se tienen en cuenta asimismo los costes adicionales asociados a la ejecución del proyecto. Como se ha tenido la fortuna de poder acceder a muchos recursos a través de internet, estos costes adicionales no son muy elevados. En estos costes entrarían:

- Realización de tutoriales (Android, SQL)
- Conjunto de SDKs y APIs utilizadas
- Conexión a internet

Cabe destacar que los tutoriales que se han realizado han sido todos de carácter gratuito. En la actualidad se tiene la suerte de disponer de infinidad de información en internet. Además, dado su carácter más dinámico, se han visto también muchos tutoriales en la plataforma YouTube.
Por otro lado, se han utilizado SDKs y APIs gratuitas en su totalidad, por lo que los costes adicionales los componen solamente la conexión a internet utilizada.

Concepto	Coste	Periodo	Coste Total
Conexión a internet	30€/mes	4 meses	120€

Tabla 3: Costes adicionales

7.3 Coste total del proyecto

Sumando los costes de recursos humanos y los costes adicionales, se obtiene el coste total del proyecto:

Concepto	Valor
Costes de recursos humanos	7.975€
Costes adicionales	120€
Coste total del proyecto	**8.095€**

Tabla 4: Coste total del proyecto

8 CONCLUSIONES

Estoy muy satisfecho con el resultado de este proyecto, puesto que me ha permitido poner en práctica muchos de los conocimientos que he adquirido durante mis estudios en la Universidad de Sevilla y en la Universidad de Florencia. También me ha ayudado mucho para darme cuenta de la dimensión que tiene un proyecto por pequeño que parezca y la cantidad de decisiones que hay que tomar a lo largo de su desarrollo.

Desde que me compré mi primer Android en el primer año de carrera me llamó la atención el desarrollo de aplicaciones móviles, por lo que cuando empecé a buscar sobre qué hacer el proyecto fin de carrera y en el tablón vi la posibilidad de realizar una aplicación móvil, no lo dude un segundo.

Este proyecto me sirvió para valerme por mi mismo a la hora de resolver todos los problemas que se me presentaron, ya que aunque toda la información se encuentra en internet, tuve que emplear bastante tiempo en discernir cuáles eran los tutoriales que de verdad iban a ayudarme y cuáles no.

Más tarde, tuve que hacer frente a una gran cantidad de decisiones y problemas que aparecieron en el desarrollo del proyecto. He de reconocer que la primera intención fue la de realizar el proyecto con otro compañero, pero ante la imposibilidad de encontrar a nadie, tuve que hacerme frente del proyecto por completo. Este gran inconveniente no me ha permitido que la primera versión de ParkApp tenga todas las características y funcionalidades que me hubiera gustado, pero aun así me siento muy orgulloso del resultado obtenido.

Sinceramente, pienso que ParkApp tiene un hueco en el mercado puesto que existen una gran cantidad de empresas involucradas en el sector. Para ello es necesario una inversión extra de tiempo y dinero como se comentó anteriormente para poder desarrollar un producto viable y definitivo.

REFERENCIAS

1 Infografía de la historia de las tiendas de aplicaciones móviles. Consulta web:

http://blog.shoutem.com/2012/02/07/infographic-the-history-of-mobile-app-stores/

2 La evolución de las aplicaciones móviles. Consulta web:

http://upsasoyyo.wordpress.com/2013/09/17/aplicaciones-moviles-la-evolucion/

3 Página oficial de Android Auto. Consulta web:

https://www.android.com/auto/

4 Aplicación Android Parkopedia Estacionamiento. Consulta web:

https://play.google.com/store/apps/details?id=com.parkopedia&hl=es

5 Aplicación Android ParkMe Parking. Consulta web:

https://play.google.com/store/apps/details?id=com.parkme.consumer&hl=es

6 Aplicación Android Parking. Consulta web:

https://play.google.com/store/apps/details?id=com.uknowapps.android.parking&hl=es

7 Aplicación Android m.Parking. Consulta web:

https://play.google.com/store/apps/details?id=lt.sisp.itero.parking.client&hl=es

8 Aplicación Android Parking Manijak. Consulta web:

https://play.google.com/store/apps/details?id=com.yuspin.android.parkingmanijak&hl=es

9 Noticia sobre apparkB. Consulta web:

http://w1.bcn.cat/barcelonablog/actualidad/una-app-para-aparcar-mejor?lang=es

10 Cuota de mercado de smartphones según sistema operativo. Consulta web:

http://www.idc.com/prodserv/smartphone-os-market-share.jsp

11 Apache vs NGINX. Consulta web:

http://www.theorganicagency.com/apache-vs-nginx-performance-comparison/

12 MariaDB vs MYSQL. Consulta web:

http://www.vozidea.com/que-es-mariadb-y-ventajas-frente-mysql

13 Tutoriales oficiales Android. Consulta web:

http://developer.android.com/

14 Tutorial Android de Manuel Muñoz Mir. Consulta web:

https://www.youtube.com/playlist?list=PLyBd7TyyK5mqpHU_F7WzIOLJ1WWc8FqCH

15 Tutorial Android de Código Alonso. Consulta web:

https://www.youtube.com/playlist?list=PL2A212B6BD6C5D596

16 Tutorial Android de thenewboston. Consulta web:

https://www.youtube.com/playlist?list=PL2F07DBCDCC01493A

17 Tutorial MySQL. Consulta web:

http://www.w3schools.com/sql/

18 Dudas varias. Consulta web:

http://stackoverflow.com/

19 Asociación Española de Aparcamientos y Garajes. Consulta web:

http://www.asesga.org/asesga.php

GLOSARIO

Activity	Actividad que puede realizar un usuario en Android.
API	Interfaz de programación de aplicaciones.
Aria	Mecanismo de almacenamiento de datos usada por defecto por el sistema administrador de bases de datos relacionales MariaDB.
Array	Contenedor de objetos de un mismo tipo en Android.
ASESGA	Asociación Española de Aparcamientos y Garajes
Fragment	Porción de la interfaz de usuario de una *activity* en Android.
GNU GPL	Licencia Pública General de GNU.
GPS	Sistema de posicionamiento global.
HTTP	Protocolo de transferencia de hipertexto.
IDC	International Data Corporation
IDE	Entorno de desarrollo interactivo.
Inflater	Encargado de ejecutar ciertos elementos en Android.
Intent	Encargado de realizar operaciones entre las *activities* en Android.
JSON	Subconjunto de la notación literal de objetos de JavaScript para el intercambio de datos.
Layout	Elemento de la interfaz de usuario en Android.
Listener	Elemento que permanece a la espera de alguna acción en Android.
MyISAM	Mecanismo de almacenamiento de datos usada por defecto por el sistema administrador de bases de datos relacionales MySQL.
PHP	Lenguaje de programación de uso general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico.
Preference	Elemento que guarda un valor incluso al cerrar la aplicación en Android.
SDK	Kit de desarrollo de software.
Toast	Mensaje que aparece momentáneamente en una aplicación en Android.
UNIX	Sistema operativo portable, multitarea y multiusuario.
XAMPP	Sistema de gestión de bases de datos MySQL, el servidor web Apache y los intérpretes para lenguajes de script: PHP y Perl.
XML	Lenguaje de marcas extensible.

ANEXO A

A.1 Clase MainActivity

```
package com.example.parkapp;

import java.util.Locale;

import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.graphics.Color;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.Circle;
import com.google.android.gms.maps.model.CircleOptions;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;

public class MainActivity extends android.support.v4.app.FragmentActivity {
      public static final double KM = 1000;
      public static final double MI = 1609.344;
      private Button btn_buscar;
      private EditText edt_dist;
      private GoogleMap map;
      private Locale miLocale;
      private Location loc;
      private LatLng posicion;
      private LatLng pos_prueba;
      private Circle circulo = null;
      private float[] distance = new float[2];
      private SharedPreferences prefs;
      private String key_uni;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        //Se busca si hay idioma guardado
        cargarLocale();
```

```
//Se lanza el layout
setContentView(R.layout.activity_main);

//Se instancian los elementos del layout
btn_buscar = (Button)findViewById(R.id.btn_buscar);
edt_dist = (EditText)findViewById(R.id.edt_dist);
map = ((SupportMapFragment) getSupportFragmentManager()
      .findFragmentById(R.id.mapview)).getMap();

//Se establecen los kilómetros como unidad predeterminado
prefs = PreferenceManager.getDefaultSharedPreferences(getBaseContext());
key_uni = prefs.getString("unidad", "KM");

//Se obtienen las coordenadas de la posición actual
loc = localizar();
posicion = new LatLng(loc.getLatitude(), loc.getLongitude());

//Se generan las coordenadas para una posición de prueba
pos_prueba = new LatLng(loc.getLatitude()+2, loc.getLongitude()+1);

//Se genera un marcador en la posición actual
final MarkerOptions marcador = new MarkerOptions();
marcador.position(posicion);
marcador.title("R.string.ubicacion");
map.addMarker(marcador);

//Se genera un marcador para la posición de prueba
final MarkerOptions prueba = new MarkerOptions();
prueba.position(pos_prueba);
prueba.title("Prueba");
map.addMarker(prueba);

//Se activa un listener para cuando se introduzca un valor
btn_buscar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
     //Se convierte la distancia introducida en una cadena
     String distancia = edt_dist.getText().toString();

     //Si no se ha introducido una distancia, se muestra un toast de error
     if (distancia.equals("")) {
           Toast.makeText(getApplicationContext(), R.string.toast_dist,
           Toast.LENGTH_LONG).show();
     }
     //Si se ha introducido algún valor, se convierte a double
     else {
           double dist = Double.parseDouble(distancia);

           //Si el valor es mayor que 0, se calcula la distancia
           if (dist > 0) {
                 //Si las unidades son kilómetros, se usa el factor KM
                 if (key_uni.equals("KM")) {
                       dist = dist*KM;
                 }
                 //Si las unidades son millas, se usa el factor MI
                 else {
                       dist = dist*MI;
                 }
```

```
                        //Si ya existe un círculo, se elimina
                        if (circulo != null) {
                              circulo.remove();
                        }

                        /*Se genera un círculo con la distancia deseada y
                          centrado en la posición actual*/
                        circulo = map.addCircle(new CircleOptions()
                        .center(posicion)
                        .radius(dist)
                        .strokeColor(Color.BLACK)
                        .fillColor(Color.TRANSPARENT));

                        //Se calcula la distancia entre los dos marcadores
                        Location.distanceBetween(marcador.getPosition().latitude,
                        marcador.getPosition().longitude,
                        prueba.getPosition().latitude,
                        prueba.getPosition().longitude, distance);

                        /*Si la distancia calculada es mayor a la introducida, se
                          muestra un toast indicando que está fuera*/
                        if (distance[0] > dist) {
                              Toast.makeText(getBaseContext(),
                              R.string.toast_fuera, Toast.LENGTH_LONG).show();
                        }
                        /*Si la distancia calculada es menor a la introducida, se
                          muestra un toast indicando que está dentro*/
                        else {
                              Toast.makeText(getBaseContext(),
                              R.string.toast_dentro, Toast.LENGTH_LONG).show();
                        }
                        //Se lanza la actividad que contiene la animación
                        Intent i = new Intent(getApplicationContext(),
                                    Descarga.class);
                        startActivity(i);
                  }
                  //Si el valor es menor o igual que 0, se muestra un toast de error
                  else {
                        Toast.makeText(getApplicationContext(),
                        R.string.toast_dist, Toast.LENGTH_LONG).show();
                  }
            }
          }
      });
}

  //Método que genera el menú de la aplicación
  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater infl = getMenuInflater();
        infl.inflate(R.menu.main, menu);

        return super.onCreateOptionsMenu(menu);
  }

   //Método con cada uno de los elementos del menú de la aplicación
  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
```

```
                /*Si la opción elegida es el mapa satélite, se cambia a dicha
                  vista y se muestra un toast personalizado*/
                case R.id.menu_sat:
                      map.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
                      textIconToast(R.layout.toast_sat, Toast.LENGTH_LONG);
                      break;
                /*Si la opción elegida es el mapa normal, se cambia a dicha
                  vista y se muestra un toast personalizado*/
                case R.id.menu_map:
                      map.setMapType(GoogleMap.MAP_TYPE_NORMAL);
                      textIconToast(R.layout.toast_map, Toast.LENGTH_LONG);
                      break;
                /*Si la opción elegida es el mapa híbrido, se cambia a dicha
                  vista y se muestra un toast personalizado*/
                case R.id.menu_hyb:
                      map.setMapType(GoogleMap.MAP_TYPE_HYBRID);
                      textIconToast(R.layout.toast_hyb, Toast.LENGTH_LONG);
                      break;
                /*Si la opción elegida es el mapa terreno, se cambia a dicha
                  vista y se muestra un toast personalizado*/
                case R.id.menu_ter:
                      map.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
                      textIconToast(R.layout.toast_ter, Toast.LENGTH_LONG);
                      break;
                //Si la opción elegida es configuración, se lanza dicha actividad
                case R.id.menu_confg:
                      Intent c = new Intent(getApplicationContext(),
                                  Confg.class);
                      startActivity(c);
                      break;
                //Si la opción elegida es información, se lanza dicha actividad
                case R.id.menu_info:
                      Intent i = new Intent(getApplicationContext(), Info.class);
                      startActivity(i);
                      break;
                //Si la opción elegida es salir, se sale de la aplicación
                case R.id.menu_salir:
                      finish();
                      break;
            }
            return super.onOptionsItemSelected(item);
    }

    //Método que genera un toast personalizado
    private void textIconToast(int layout, int duration) {
      LayoutInflater inflater = getLayoutInflater();
      View layoutView = inflater.inflate(layout, null);
      Toast toast = new Toast(getApplicationContext());
      toast.setDuration(duration);
      toast.setView(layoutView);
      toast.show();
  }

  //Método que lee desde las preferencias
  public static String leePreferencias(String key, Context context) {
      SharedPreferences preferences = PreferenceManager
                                      .getDefaultSharedPreferences(context);
      return preferences.getString(key, "");
  }
```

```java
    //Método que carga el idioma desde las preferencias
    public void cargarLocale() {
        String PrefIdioma = "idioma";
        String idioma = leePreferencias (PrefIdioma, getBaseContext());
        cambiarLocale(idioma);
    }

    //Método que cambia el idioma de la aplicación
    public void cambiarLocale(String locale) {

        //Si está vacío, queda el idioma por defecto del dispositivo
        if (locale==="")
            return;

        //Se configura el idioma cargado desde SharedPreferences
        miLocale = new Locale(locale);
        Locale.setDefault(miLocale);
        android.content.res.Configuration config = new Android
                                                    .content.res.Configuration();
        config.locale = miLocale;
        getApplicationContext().getResources().updateConfiguration(config,
        getApplicationContext().getResources().getDisplayMetrics());
    }

    //Método que devuelve la localización actual
    private Location localizar() {
         LocationManager locManager;
         LocationListener locListener;
         final Location loc;

         //Se obtiene una referencia al LocationManager
         locManager = (LocationManager)getSystemService(Context.LOCATION_SERVICE);

         //Se obtiene la última posición conocida
         loc = locManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);

         //Se registra para recibir actualizaciones de la posición
         locListener = new LocationListener() {
              public void onLocationChanged(Location location) {
              }
              public void onProviderDisabled(String provider){
              }
              public void onProviderEnabled(String provider){
              }
              public void onStatusChanged(String provider, int status, Bundle extras){
              }
         };

       //Se configuran las opciones para actualizar la posición
        locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 30000, 0,
                                          locListener);

        return loc;
    }

}
```

A.2 Clase Descarga

```
package com.example.parkapp;

import android.app.Activity;
import android.graphics.drawable.AnimationDrawable;
import android.os.Bundle;
import android.widget.ImageView;

public class Descarga extends Activity {
    private ImageView ianimacion;
    private AnimationDrawable animacion;

      @Override
      protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.descarga);

            //Se genera la animación y se inicia
            ianimacion = (ImageView)findViewById(R.id.animacion);
            ianimacion.setBackgroundResource(R.drawable.animacion);
            animacion = (AnimationDrawable)ianimacion.getBackground();
            animacion.start();
      }
}
```

A.3 Clases Lista, Parking y ParkingAdapter

A.3.1 Clase Lista

```
package com.example.parkapp;

import java.util.ArrayList;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ListView;

public class Lista extends Activity{
      ListView listaParking;
      private ArrayList<Parking> parking;
      private ParkingAdapter adapter;

      @Override
      protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.lista);

            //Se instancia la lista
            listaParking = (ListView)findViewById(R.id.lista);

            //Se genera el array
            parking = new ArrayList<Parking>();

            //Se rellena el array con datos de prueba
            rellenarArrayList();
```

```
            //Se genera el adaptador para el array con la lista de parkings
            adapter = new ParkingAdapter(getApplicationContext(), parking);
                                        listaParking.setAdapter(adapter);
        }

        //Método que rellena el array con datos de prueba
        private void rellenarArrayList() {
            parking.add(new Parking("Parking Pepe","2km","200","9:00-18:00",
                        "0.0135€"));
            parking.add(new Parking("Parking Manolo","3.5km","300","10:00-18:00",
                        "0.015€"));
            parking.add(new Parking("Parking Ramon","4km","250","9:00-19:00",
                        "0.0122€"));
            parking.add(new Parking("Parking Raul","5.5km","400","10:00-20:00",
                        "0.0136€"));
            parking.add(new Parking("Parking Pedro","6km","150","9:00-20:00",
                        "0.0141€"));
        }
}
```

A.3.2 Clase Parking

```
package com.example.parkapp;

//Clase con los distintos métodos para trabajar con el tipo Parking
public class Parking {
        private String nombre, distancia, plazas, horario, precio;

        public Parking(String nombre, String distancia, String plazas, String horario,
                        String precio) {
            this.nombre = nombre;
            this.distancia = distancia;
            this.plazas = plazas;
            this.horario = horario;
            this.precio = precio;
        }

        public String getNombre() {
            return nombre;
        }

        public void setNombre(String nombre) {
            this.nombre = nombre;
        }
        public String getDistancia() {
            return distancia;
        }

        public void setDistancia(String distancia) {
            this.distancia = distancia;
        }

        public String getPlazas() {
            return plazas;
        }

        public void setPlazas(String plazas) {
            this.plazas = plazas;
        }
```

```
	public String getHorario() {
		return horario;
	}

	public void setHorario(String horario) {
		this.horario = horario;
	}

	public String getPrecio() {
		return precio;
	}

	public void setPrecio(String precio) {
		this.precio = precio;
	}
}
```

A.3.3 Clase ParkingAdapter

```
package com.example.parkapp;

import java.util.ArrayList;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.TextView;

public class ParkingAdapter extends ArrayAdapter<Parking> {
	private Context context;
	private ArrayList<Parking> datos;

	//Se genera el adaptador
	public ParkingAdapter(Context context, ArrayList<Parking> datos) {
		super(context, R.layout.layout_adaptador, datos);

		// Se guardan los parámetros en variables de clase
		this.context = context;
		this.datos = datos;
	}

	@Override
	public View getView(int position, View convertView, ViewGroup parent) {
		//Se genera y se lanza el layout del adaptador
		LayoutInflater inflater = LayoutInflater.from(context);
		View item = inflater.inflate(R.layout.layout_adaptador, null);

		//Se instancia y se rellena cada uno de los campos del adaptador
		TextView nombre = (TextView)item.findViewById(R.id.txt_name);
		nombre.setText(datos.get(position).getNombre());
		TextView distancia = (TextView)item.findViewById(R.id.txt_dist);
		distancia.setText(datos.get(position).getDistancia());
		TextView plazas = (TextView)item.findViewById(R.id.txt_plazas);
		plazas.setText(datos.get(position).getPlazas());
		TextView horario = (TextView)item.findViewById(R.id.txt_horario);
		horario.setText(datos.get(position).getHorario());
		TextView precio = (TextView)item.findViewById(R.id.txt_precio);
```

```
            precio.setText(datos.get(position).getPrecio());

            return item;
        }
}
```

A.4 Clases Conf y Preferencias

A.4.1 Clase Conf

```
package com.example.parkapp;

import android.app.Activity;
import android.os.Bundle;

//Clase que realiza la transación de una actividad a un fragment
public class Confg extends Activity {

        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.confg);
            getFragmentManager().beginTransaction().replace(R.id.confg,
            new Preferencias()).commit();
        }
}
```

A.4.2 Clase Preferencias

```
package com.example.parkapp;

import android.content.Intent;
import android.content.SharedPreferences;
import android.content.SharedPreferences.OnSharedPreferenceChangeListener;
import android.os.Bundle;
import android.preference.PreferenceFragment;
import android.widget.Toast;

public class Preferencias extends PreferenceFragment implements
OnSharedPreferenceChangeListener {
        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            //Se obtienen las preferencias desde el archivo xml
            addPreferencesFromResource(R.xml.preferencias);

            //Se genera la pantalla descrita en dicho archivo xml
            this.getPreferenceScreen().getSharedPreferences()
            .registerOnSharedPreferenceChangeListener(this);
        }

       //Método que muestra las distintas opciones de las preferencias
        @Override
        public void onSharedPreferenceChanged(SharedPreferences sharedPreferences,
                                              String key) {
```

```
            /*Si la opción elegida es idioma, se vuelve a la actividad principal
              para que se realicen los cambios*/
            if (key.equals("idioma")) {
                  Intent i = new Intent(getActivity(), MainActivity.class);
                  i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                  startActivity(i);
            }
            /*Si la opción elegida es unidad, se vuelve a la actividad principal
              para que se realicen los cambios y se muestra un toast con la unidad*/
            if (key.equals("unidad")) {
                  Toast.makeText(getActivity(), R.string.toast_unidad,
                                 Toast.LENGTH_LONG).show();
                  Intent i = new Intent(getActivity(), MainActivity.class);
                  i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                  startActivity(i);
            }
    }
}
```

A.5 Clase Info

```
package com.example.parkapp;

import android.app.Activity;
import android.os.Bundle;

//Clase que lanza el layout con la información sobre la aplicación
public class Info extends Activity {

      @Override
      protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.info);
      }
}
```